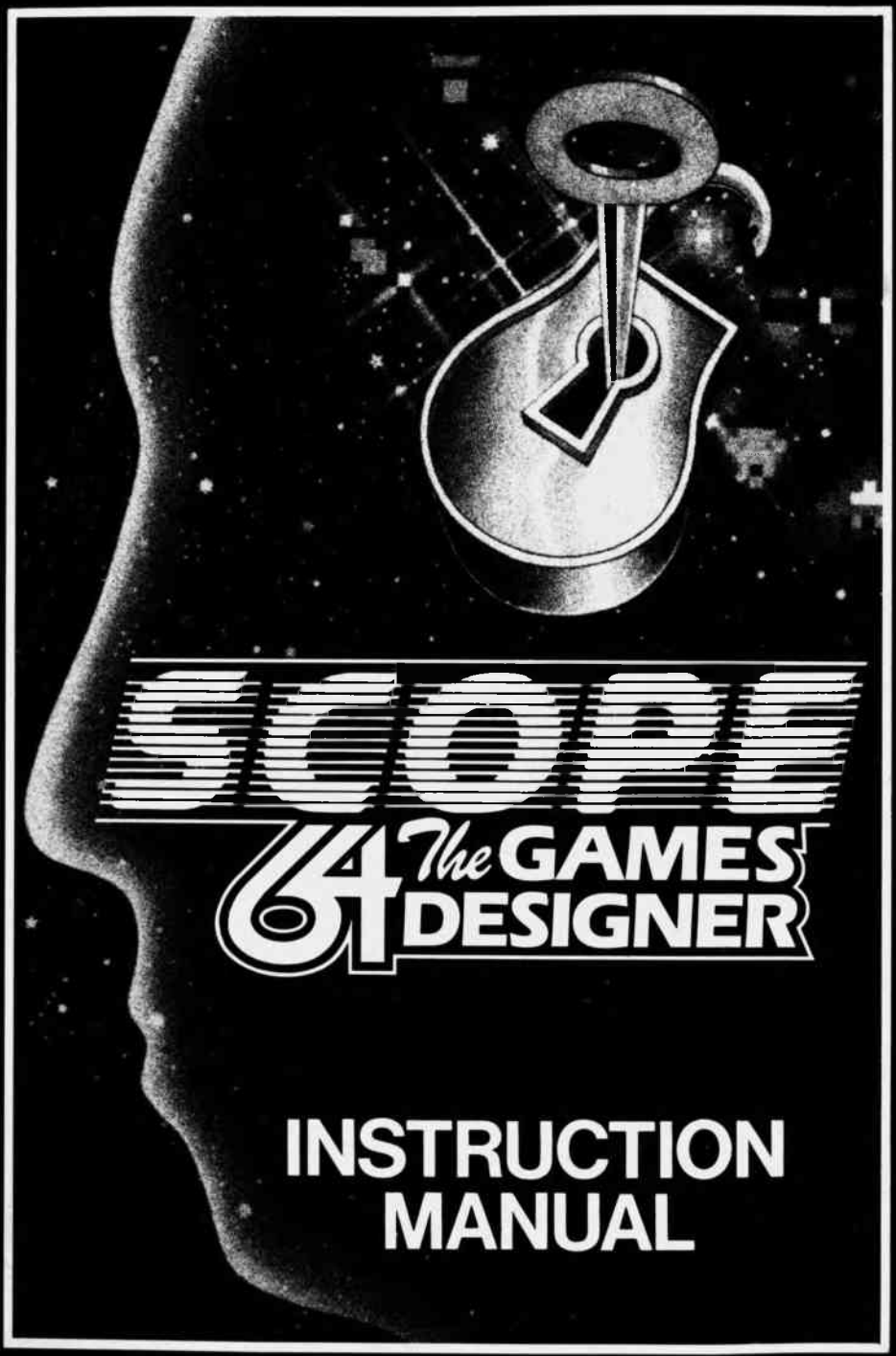

SCORPI
64
The GAMES DESIGNER
INSTRUCTION MANUAL

Program by Allen E Pendle

Manual by ISP Marketing Limited
Copyright ISP Marketing Limited

Introduction

SCOPE 64 is a fully structurable language, primarily intended for high speed handling of graphics, sprites, colour, sound, music and animation.
It consists of 46 command words which are written in BASIC Rem statements. The program is then compiled or re-written into an area of memory reserved for it, and run by means of a SYS command.

Although many SCOPE words only handle integer numbers in the range 0-255, there are also several words which enable the programmer to deal with double byte integers (0-65535).

The handling of colour, sound and music is approached in a different manner then in BASIC. See the relevant section of this manual for full details.
As far as possible SCOPE words have been made to appear similar to their BASIC counterparts. This is to aid programmers, most of whom will already be familiar with BASIC.

What the words achieve, however, is much more powerful as the instructions are compiled into machine code with the resultant increase in the speed of execution.

SCOPE also has many other advantages in its flexibility and ability to construct almost any form of structure in its programming. It should be regarded as a series of programming building blocks enabling the functions of the machine to be used simply and effectively.

The code generated by the SCOPE compiler is *NOT* dependent upon SCOPE being resident in high memory. Instructions are included in the manual to save SCOPE compiled code which is then a machine code program in its own right which is run by a SYS call to its ORG address (See section 1).

In the following pages the language and its operation will be explained step by step. Throughout this manual we have had to assume that the programmer has at least a working knowledge of BASIC and a simple understanding of the normal operations of the computer.

Loading Scope and the Demonstration Programs

To load SCOPE type the following as a direct statement:-

LOAD"SCOPE"

When you receive the message that SCOPE has been found press the space key as usual. When you receive the message that SCOPE has loaded type as a direct statement – RUN.

SCOPE will now be automatically loaded into high memory and when completed you will be shown the title page. When you press any key as requested the display will be Lt. green on black. Resetting the system will restore the normal screen colours if you so wish.

To load the Demonstration routines.

These are on the reverse side of your tape.
Enter the following as a direct statement and when loaded, compile (SYS 49583) and then run (SYS 15500).

The only exception to this is the program "HORSES", which loads the sprite data for "ASCOT". You should RUN the program "HORSES" and then load "ASCOT" and proceed as usual.

List of Demonstration Routines

1) LOAD "SHOOT IN"
2) LOAD "WMUSIC"
3) LOAD "HORSES" and then "ASCOT" (See above).
4) LOAD "MOZART"
5) LOAD "MOSAIC"
6) LOAD "BACH"
7) LOAD "SPACE SOUND"

Syntax

Syntax in SCOPE is quite simple and follows the following rules:-

1) All words must be typed in BASIC REM statements.
2) All words are immediately followed by a semi-colon.
3) Commas are used between all operands.
4) Statements must end with a colon, except those that have no operands which will end in a semi-colon.
5) Words should be typed as shown in the manual and Dictionary.

Errors will give the error report 'Syntax Error' *and the line number.*

Whilst every care has been given to error checking remember that if you ask the compiler correctly to do something incorrect it will attempt to do so and may on rare occasions crash.

It is therefore recommended that you save a copy of the program before compiling, particularly if it is a long program.

Memory organisation

Before you start writing a SCOPE program you must nominate an area where the compiled SCOPE program is to be written. You must also nominate an area where SCOPE routines should be written, because SCOPE ensures that any routines are written in a special area.

After SCOPE has been loaded, the memory addresses from 2048 to 39,999 are available to the user for programming. SCOPE words are typed in BASIC Rem statements and therefore a SCOPE program must start with the word ORG.

```
10 REM ORG;15500,25000:
```

The above program line actually means:-

Start compiling SCOPE at memory location 15500 and any SCOPE routines at memory location 25000. Therefore you will have allowed yourself approximately 13k of memory for BASIC (from 2048 to 15500), 10k for SCOPE statements (15500 to 25000) and 15k for SCOPE routines (25000 to 39999).

These parameters are totally under your control and can be altered any time you wish. You may also wish to hold other data such as sprite definitions or an alternative character set somewhere towards the end of your BASIC area.

For instance a recommended area to store an alternative character set is from memory location 12288 to 14335. You may also wish to store 8 sprite definitions from memory location 14336 to 14847. In this case you would obviously wish to protect this data from being overwritten by any overflow from the BASIC area. To achieve this you use the word:-

```
20 REM ALTER; 12287:
```

This makes the computer think it has run out of memory when it reaches memory location 12287.

Let us assume that you wish to organise your memory as follows:-

BASIC AREA 2048-12287
DATA AREA 12288-14335
SCOPE AREA 15550-29999
ROUTINE AREA 30000-39999

This would be achieved by the following SCOPE program:-

```
10 REM ORG;15500,30000:
20 REM ALTER;12287:
9999 REM EXIT;
```

The last word EXIT enables you to return to BASIC after the program has compiled or after it has run. The words ORG and ALTER should always commence a SCOPE program and it should always finish with the word EXIT.

Sometimes it is useful to know how much of any area you have used. This can easily be done.

To find out how much of the BASIC area has been used type as a direct statement:-

PRINT PEEK(121)+256*PEEK(122)
and press RETURN

The last BASIC address used will be printed on the screen.

To find out how far you have progressed in the SCOPE program area, type in as a direct statement:-

PRINT PEEK(251)+256*PEEK(252)
and press RETURN

This will print on the screen the next location where any SCOPE programs will be written.

To find out how far you have progressed in the SCOPE routines area enter as a direct statement:-

PRINT PEEK(49522)+256*PEEK(49523)

This will print to the screen the next location where any SCOPE routines will be written.

It is suggested to start off with that you use 15500 for the SCOPE program area and 20000 for the SCOPE routine area. If you wish to protect your SCOPE program from any overflow from BASIC then use ALTER;14999:

You will have noticed that in the examples; the SCOPE main program area has always been shown as starting at 15500. This because the word EXIT also writes some code to the area 15000-15499.

This code is to support three special SCOPE words. These words will not be used in many programs that you write.

They are

1) **TCO;**
2) **SCR;**
3) **#MUSIC;**

If these words are not in your program you may set the Scope Program area anywhere you wish. If these words are used however you must set the area from 15500.

It follows therefore that ALTER should specify either 12287 or 14999; if these words are not used you can extend your basic area accordingly.

Compiling and Running a Scope Program

When a series of command words has been written as a SCOPE program it has to be compiled into machine code and re-written into the SCOPE program area.

To do this, type as a direct statement:-

SYS 49583
and press ENTER

You will receive either an error report if you have made a mistake, or a number will appear on the screen. This number is the next address for any further SCOPE program. There is a very good reason for this being the next rather than the last memory location used.

You may wish to use SCOPE to write some machine code routines which can be called from within a BASIC program. You will need to know therefore the start number of any routine so that you may use a SYS call from BASIC.

Used in this fashion SCOPE is a unique assembler using plain word mnemonics and enables you to write fast and powerful graphics routines without any actual knowledge of Assembler or the complexities of machine code.

To run your program when it has been compiled you use the SYS command together with the first ORG address. For example:-

SYS 15500

Saving your compiled Scope Code

On your cassette tape after the main SCOPE program, is a program called "TAPESAVE". This should be loaded by;

LOAD "TAPESAVE"
and then RUN

You will be asked for a filename (not exceeding ten letters) and the start address and finish address.

Remember that the start address will be 15000 (not 15500) if the three special words are used or the actual ORG address if not.

The finish address is the next address to be used in the routine area which you find by the direct statement PRINT PEEK (49522) +256*PEEK(49523).
To ensure that you have to save as few bytes as possible you may need to adjust your memory organisation after you have done your first compilation.

For example let us assume that you have used the ORG statement;
10 REM ORG;15500,20000:

When you compile you find that the next main area address is 15839. This means that you have used 338 bytes and you can now adjust your ORG statement to;

10 REM ORG;15500,16000:

This will leave a gap of 100 bytes between the end of the main area and the beginning of the routine area. This is always advisable. Now compile again and find out the finish address and you are ready to save your code.

Handling Numbers

SCOPE does not have any string handling facilities so all variables are numeric.
Two types of variables are offered by SCOPE:-

A) Single byte variables-where VAR is an integer variable with a value between 0 and 255.
B) Double byte variables – where DVAR is an integer variable between 0 and 65535.

There are 52 possible variables specified by the 26 letters of the alphabet, and the presence or absence of a hash symbol (#).

e.g. VAR – A to Z
VAR – #A to #Z

There are only 26 DVAR's using the letters A to Z.

Variables are very useful in a SCOPE program and are used in a similar manner as in a BASIC program. Some possible uses of variables are as follows:-

D) As fluctuating co-ordinates for characters or sprites on the screen.
B) As a medium for testing conditions.
C) As a medium for reading the keyboard.
D) As loop counters.
E) To vary the duration of pauses.
F) To vary the colours of the border, background, characters or sprites.

DVAR's are useful in real time applications, score keeping and displaying numbers to the screen, as well as longer loop counters.
They are also of great importance as address pointers and this will be explained in detail later in this manual.

Here are eight SCOPE words that deal with variables and numbers together with examples.

1) **VAR;** *a letter, a letter or a variable:* This sets a variable to a value.
2) **INC;** *a Var, a number or a Var:* Increases a variable by a value.
3) **DEC;** *a Var, a number or a Var:* Decreases a variable by a value.

Examples:

20 REM VAR;A,1: Sets Var a to value 1.
30 REM INC;B,A: Increases a Var by the value of another Var.
40 REM DEC;B,1: Decreases a Var by a value.
Similarly you can set a Var to the value of another Var as in

50 REM VAR;A,B:

We now move on to DVAR's.

4) **DVAR;** *a letter, a number:* This sets a Dvar to a value
5) **ADD;** *a Dvar, a number or a Var:* This increases a Dvar by a number from 0-255 only.
6) **MINUS;** *a Dvar, a number or a Var:* This decreases a Dvar by a number from 0-255.

Examples:
10 REM DVAR;A,20000: Sets Dvar to 20000.
20 REM ADD;A,200: Increases the Dvar by 200.
30 REM MINUS;A,200: Decreases the Dvar by 200.

Note that the increment or decrement value can be expressed as a Var.

10 REM VAR;A,200: Sets Var;A to 200.
20 REM DVAR;A,20000: Sets Dvar;A to 20000.
30 REM MINUS;A,A: Decreases Dvar;A by the value of Var;A (200).

Note
10 REM DVAR;A,A: *is not allowed*

A Dvar can only be set to a numerical value.

A special word exists for DVAR's only:-
7) **NUM;** *a number, a number,* a number or Var,a Dvar:

This word Prints on the screen the value of the Dvar specified in the 4th operand, at the line and column numbers specified in the first two operands, in the colour specified in the third operand.

Examples:

10 REM DVAR;A,20000:
20 REM NUM;10,10,0,A:

This will print the value of Dvar;A (20000) at line 10, column 10, in colour 0 (black).

or
10 REM DVAR;A,20000:
20 REM VAR;C,0:
30 REM NUM;10,10,C,A:

This does exactly the same, we hope you can see why.

The last word in this section deals with random numbers, which are a must for writing games.

7) **RAND;** *a Var, a number, a number.*

This word puts into the variable specified in the first operand, a random number between 0 and one less than the number specified in the second operand, modified or offset by the number in the third operand. The following examples will we hope make the position clear.

10 REM RAND;A,10,0: This puts into Var;A a random number between 0-9.

or

20 REM RAND;A,12,5: This puts into Var;a a random number between 5-16.

or

30 REM RAND;A,3,200: This puts into Var;A a random number between 200-202.

We can now move on to the handling of colour and graphics with some examples and some very simple programs for you to try out.

Colour Handling and Graphics

Throughout SCOPE colours are referred to by numbers from the following table:-

Colour no.	*Colour*	*Colour no.*	*Colour*	*Colour no.*	*Colour*
0	BLACK	6	BLUE	12	GREY 2
1	WHITE	7	YELLOW	13	LT. GREEN
2	RED	8	ORANGE	14	LT. BLUE
3	CYAN	9	BROWN	15	GREY 3
4	PURPLE	10	LT. RED		
5	GREEN	11	GREY 1		

WORDS USING COLOUR

1) **BDR;** *a number or a Var:* Sets the colour of the border
2) **BACK;** *a number or a Var:* Sets the colour of the background

Examples:
10 REM BDR;0: Sets the border colour to Black
20 REM BACK;8: Sets the background colour to Orange.

or
10 REM VAR;A,1:
20 REM VAR;B,10:
30 REM BDR;A:
40 REM BACK;B:

This sets the border to White and the background to Lt. Red.
The next two words deal with the 3 graphics modes available on the COMMODORE 64 and special colour handling.

3) **GMODE;** *a number:* Sets up the current Graphics mode.
There are three graphics modes that can be used.

A) STANDARD CHARACTER MODE
This is the mode in operation when you first turn on the machine. Characters are normally taken from RAM. You can however create your own character sets in RAM. See section of Commodore Programmes Reference Guide.

B) MULTI COLOUR MODE
In this mode the character data is interpreted by the Processor in a different manner. Normally a character has 64 dots in character memory and each dot can have two possible values, 1 for on and 0 for off. If the dot is on it is coloured with the character colour you chose for that screen position. If the dot is off it assumes the background colour of the screen.

However, in Multi colour mode each dot can be in one of four different colours.
1) The screen background colour.
2) Character colour 1.
3) Character colour 2.
4) Character colour 3.

The only sacrifice is that each dot is now twice as wide as before in Standard character mode.
The word to set up these colours in SCOPE is COL.

C) EXTENDED BACKGROUND COLOUR MODE
This gives you control over the background colour of each character as well as foreground colour. However, you can only display 64 characters out of your normal set of 256 so it is rather limiting and is generally used less than the other modes.

The word COL is formatted as follows:-

4) **COL;** *a number, a number, a number:*
The first operand sets character colour 1, the second operand sets character colour 2 and the third operand sets character colour 3.

Therefore to set up Multi colour mode you would use:-

```
10 REM GMODE;1:
20 REM COL;0,2,13:
```

Character colour 1 is now black, character colour 2 is now red and character colour 3 is now Lt. green.

To return to normal you would add the following:-
```
30 REM GMODE;0:
```

Note
Scope does not have any words to cover bit mapping.

Printing Characters to the Screen

The SCOPE word to use is:-
5) **PUT;** *a number or a Var., a number or a Var., a number or a Var., "Text":*
A Put statement is interpreted differently according to which Graphic Mode you are using. To start off with let us remain in Gmode;0.

The first operand is the line number.
The second operand is the column number.
The third operand is the colour number.
The text required is placed in quote marks (inverted commas).

Examples:

10 REM PUT;10,10,5,"SCOPE":

This prints the word "SCOPE" at co-ordinates 10,10 on the screen in purple.
10 REM VAR;A,10:
20 REM PUT;A,A,A, "SCOPE":

This prints the word "Scope" at co-ordinates 10,10 on the screen in Lt. red.
We will now move on to Gmode;1. When in this mode you can choose whether to display as in Gmode;0, as well as Gmode;1. For example:-
Type in the following program:-

10 REM ORG;15500,20000:
20 REM GMODE;1:
30 REM PUT;10,10,5, "SCOPE":
40 REM EXIT;

NOW COMPILE – SYS 49583.
AND RUN – SYS 15500.

As you can see this prints the word SCOPE in green as if you were in Gmode;0.

Now change line 30 to read:-

30 REM PUT;10,10,8, "SCOPE":

Now compile it and run it.

As you can see everything on the screen is now interpreted in Multi colour mode. This means that you must use colours 0 to 7 only to give normal printing when in Multi colour mode. Any other colour number will result in the characters being interpreted in Multi colour mode. The colours for Multi colour mode are specified in SCOPE using the word COL.

Extended Background Mode

To achieve the different colours in extended background mode use either a shifted character or a 'Reverse On' control character in the text, or both.

Programming Aids

This section of the manual deals with the following:-

A) Structure.

B) Conditions

C) Useful extras

STRUCTURE

The first SCOPE word in this section is

1) **LABEL;** *a letter:*

Labels are reference points which are essential as SCOPE does not have any means of jumping or calling specified line number. There are 104 possible labels which may be used.

A – Z
#A – #Z
$A – $Z
%A – %Z

As an aid to indentification to what a label signifies the word NOTE may be used, which is really the equivalent of the BASIC Rem statement.

2) **NOTE;** *text:*

You will notice that in this case inverted commas are not necessary. This word tells the processor to ignore anything following until it reaches a colon. If the colon is omitted then a syntax error message will stop compilation and be displayed.

One of the great advantages of SCOPE is the ability to define a routine, which is either self sufficient or can itself make reference to other routines.
This enables the programmer to build up a program in blocks, testing each part as he goes. The final program could then be a single call. This will become apparent as we progress.

3) **RTN;**

This word tells the processor to write any following code in the routine area specified in the initial ORG statement until it encounters an END statement.

4) **END;**

This word tells the processor to write future code in the main SCOPE program area.

You now need to be able to jump to a specified label or call a specified routine identified by a label.

5) **JUMP;** *a label:*

6) **CALL;** *a label:*

The first word JUMP is the equivalent to the BASIC Goto statement. It tells the processor to jump to the label specified.

The second word CALL tells the processor to call a routine specified by a label and on return to continue processing at the next line of the program.

You must not attempt to JUMP to a routine, nor must you CALL a label that is not a routine. For example:-

```
10 REM LABEL;A:
20 REM PUT;10,10,0, "SCOPE":
30 REM JUMP;A:
```

This is a continuous loop!

```
10 REM RTN;
20 REM LABEL;A:
30 REM PUT;10,10,0, "SCOPE":
40 REM END;
50 REM CALL;A:
```

This writes a routine and then calls it.

To take advantage of the structuring abilities of SCOPE one needs to be able to test for conditions, and the following word does just that.

7) **TEST;** *a Var., a number or a Var., a test number, a label:*

This works as follows:-
The first operand is the variable to be tested.
The second operand is the value against which it is tested.
The third operand is a test number which signifies the condition under comparison. (see Table.)
The fourth operand is the label from which processing will continue if the condition is satisfied.

The operating system automatically computes whether the label is in the routine area or the main program area and writes a jump or a call accordingly.

Table of tests

208 – Jump or call if number tested is equal to the Var.
240 – Jump or call if number tested is not equal to the Var.
176 – Jump or call if number tested is greater than the Var.
144 – Jump or call if number tested is less than or equal to the Var.

If you have a variable X and a number Y then the conditions would be as follows:

208 = X=Y
240 = X<>Y
176 = X<Y
144 = X>=Y

Examples:

```
10 REM VAR;A,10:
20 REM TEST;A,11,240,B:
```

This would call or jump to the label B because A is not equal to 11.

10 REM VAR;A,10:
20 REM TEST;A,10,208,B:
This will call or jump to label B because A=10.

10 REM VAR;A,10.
20 REM TEST;A,9,144,B:
This will call or jump to label B because 9, the number tested is smaller than the value in Var.A.

10 REM VAR;A,10:
20 REM TEST;A,11,176,B:
This will call or jump to the label B because 11, the number tested is larger than the value in Var.A.

It is worthwhile noting that this test would have succeeded also if the number tested had equalled Var.A.

Note
The one test that the processor will not pick up correctly is if you try to use TEST to call one routine from within another. To do this you should ensure that the test refers to a label within the routine on one condition and use the word CALL if the condition is not satisfied.

Example:

10 REM RTN;
20 REM LABEL;A
30 REM TEST;X,65,240,B:
40 REM CALL;C:
50 REM LABEL ;B:
60 REM END;
This routine will call routine C only if Var X = 65

A Var. may be tested not only against a number, but also against another Var.
These powerful commands enable almost any type of structures to be assembled. Fixed structured languages such as Pascal, and Structured BASIC have structures such as:-

IF THEN ELSE
REPEAT UNTIL
and so on.

By combining tests one can acheive equivalents and many others without being restricted.

SCOPE also has the ability to test a Dvar, using the word LIM, which is an abbreviation of limit.

8) **LIM;** *a Dvar, a number, a label:*

The first operand is the Dvar under test, the second operand is the value against which it is tested. The third operand is the label from which program execution will continue should the condition be satisfied.

Again the operating program checks whether the label is in the routine area or the main SCOPE program aea, and writes the appropriate code.

Useful EXTRAS

It is obviously essential to have the ability to clear the screen. SCOPE handles this function with the word WIPE which has no operands.

1) **WIPE;**

Due to the sheer speed of machine code execution made possible by SCOPE it is often necessary to slow down operations by means of a delay.

2) **PAUSE;** *a number or a Var:*

The operand may be any value between 1 and 255, a value of 0 would give the same delay in fact as a value of 255. This word does not work from the Jiffy clock, but counts cycles. Therefore each value causes a delay of roughly 1/500th of a second, the value 255 giving a delay of roughly 1/2 a second.

Two special words enable you to display the screen in more than one background colour. You have two colours but split the screen into either two or three areas.

These words utilise the interrupt facilities of the machine and uses the number of raster lines to set the areas of colour. We will not attempt to go into detail about raster lines. Suffice it to say that there are 262 of them but only 199 are within the visible area of the screen.

Raster lines 0 to 49 are not visible.
Raster lines 50 to 249 are visible.
Raster lines 250 to 262 are not visible.

3) **TCO;** *a number, a number, a number, a number:*

The first operand refers to the first raster line where you wish the colour change to take place.
The second operand refers to the second raster line where you wish the colour to change again.
The third operand specifies the first colour.
The fourth operand specifies the second colour.

4) **UCO;**

This word has no operands but disables the split screen set up by TCO.

The next two words are more than useful extras; they provide enormous programming power. They are the equivalents of Peek and Poke in BASIC.

The words are:-

5) **MEM;** *a Dvar, a number or a Var.*
The Dvar is here used as a memory address pointer, and the second operand can only be a number between 0 and 255.

```
10 REM DVAR;A,30000:
20 REM MEM;A,20:
```

This will poke into memory location 30000 the value 20.

```
10 REM DVAR;A,30000:
20 REM VAR;B,15:
30 REM MEM;A,B:
```

This will poke memory location 30000 with the value 15.

6) **VIEW;** *a Dvar, a Var:*

The Dvar is again used as a memory address pointer and the Var is where the value of the contents of that location will be placed.

Try the following small program.

```
10 REM ORG;15500,20000:
20 REM VAR;A,0:
30 REM DVAR;B,30000:
40 REM MEM;B,123:
50 REM VIEW;B,A:
60 REM EXIT;
```

Now compile. (SYS 49583)
and run (SYS 15500)

Now to see the value present in Var.A enter as a direct statement;

PRINT PEEK (49308) and RETURN

The result, of course should be 123.

This word is therefore the equivalent of the BASIC Peek function.

Reading the Keyboard

1) **GET;** *a Var:*

This word reads the ASCII value of any key pressed on the keyboard and puts this value into the specified variable. If no key is pressed, the value 0 is placed in the variable.

A simple keyboard key-scan routine in SCOPE would be as follows.

```
10 REM GET;K:
20 REM TEST;K,65,240,A:
```

This will read the keyboard and continue processing the program from Label;A if no key is pressed, or if any other key than 'A' is pressed.

Scrolling

SCOPE includes a command, which deals with scrolling the entire screen by eight pixels or one character at a time, in any of four directions.

1) **SCR;** *a number:*

The number refers to the direction and must be:

0 = right
1 = left
2 = up
3 = down

Any number greater than 3 will be interpreted as a down instruction.

Although these words will give you scrolling of the whole screen in any direction, flickering will be seen at the screen edges where you are printing the new data to be scrolled. To overcome this effect you use the followisng word.

3) **XBDR;** *a number:*

where 0 = left or right.
1 = up or down.

This word shrinks the visible screen display at either the left/right edges or at the top/bottom edges, thus masking the printing of any data being printed.
Note that XBDR used once will shrink the screen and used again will return the screen to normal.

Sprite Graphics

Sprites are special user definable characters which can be displayed anywhere on the screen. They are supported by the VIC-II chip and as such are indpendent of other graphics. They can be defined in standard form (2 colours) or in multicoloured form (four colours).

Up to eight sprites can be displayed and moved on the screen at any one time.

Other special features include:-

A) 24 dot horizontal by 21 dot vertical size.

B) Individual colour control for each sprite.

C) Magnification (x2) in either or both directions.

D) Control over sprite/background priority.

E) Fixed sprite to sprite priority.

F) Collision detection sprite to sprite OR sprite to background.

Further details are given on pages 131 to 145 of the Commodore Programmers Reference Guide.

Sprite handling in SCOPE is achieved with just four command words.

1) **SPRTS;** *a number, a number, a number, a number:*

This word sets up various parmeters for any or all of the sprites you wish to use.

The five operands are as follows:-

1) This number between 1-255 enables the various sprites according to Table A.
2) This number between 1-255 sets the mode for the various sprites as per Table A.
3) This number between 1-255 sets the vertical size for the various sprites as per Table A.
4) This number between 1-255 sets the horizontal size for the various sprites as per Table A.
5) This number between 1-255 sets the sprite to background priority of the various sprites as per Table A.

Table A

Sprite Number	*Number to be used*
0	1
1	2
2	4
3	8
4	16
5	32
6	64
7	128

Any combination of these numbers can be used, to affect the particular choice of sprites you wish. Default conditions are sprites disabled-background priority over sprite.

Look at the following examples:-

10 REM SPRTS;255,0,0,0,48:

This enables all sprites in standard mode, in normal size, and sprites 4 & 5 will have priority over background.

10 REM SPRTS;7,1,4,0,2:

This enables sprites 0,1 & 2; Sprite 0 will be in Multi colour mode; Sprite 2 will be vertically magnified and sprite 1 will have priority over background.

10 REM SPRTS; 31,16,31,31,8:

This enables sprites 0,1,2,3, & 4; Sprite 4 will be in multi-colour mode; all five sprites will be magnified in both directions and sprite 3 will have priority over background.

It should be noted that sprite to sprite priorities are fixed. Sprite 0 has the highest priority and will always appear to pass in front of all other sprites. Sprite 7 on the other hand has the lowest priority and will allways appear to pass behind all other sprites.

Before we go on to the other words it should be noted that from now on all words have to be proceeded with a has symbol (#). With the exception of SPRTS all words that handle sprites or sound and music are Hash words. This is fairly easy to remember.

2) **#BLK;** *a number, a number:*

The first operand is the sprite number (0-7).
The second operand is a single byte number which is a pointer to where the sprite shape data is located.

Although sprite data can be held almost anywhere in memory it is advisable at first to choose one of the following locations.
1) If you have more than 3 sprite definitions to use then you can use the cassette buffer area, from memory location 832 to 1022.
2) If you have more than 3 sprite definitions then use from memory location 12288 onwards.
3) If you have more than 3 sprite definitions and are also using an alternative character set then use from memory location 14336 onwards.

To arrive at the 2nd number in the word BLK; divide the memory location of the start of the chosen area for the sprite definition by 64.

For example, the first definition in the cassette buffer area will be at BLK;13: the next at BLK;14: the next at BLK;15:

If you are starting your definitions at memory location 12288 your first block number will be at BLK;192: and so on.
If you are starting your definitions at 14336 then your first block number will be at BLK;224: and so on.

Example:

10 REM #BLK;0,13:

This tells the processor that the data for sprite 0 is to be taken from the 64 bytes starting at memory location 832.

20 REM #BLK;6,192:

This tells the processor that the data for sprite 6 is to be taken from the 64 bytes starting at memory location 12288.

This use of pointers to sprite data provides an economical and powerful aid to true animation. If you examine the listing of the "ASCOT" demonstration program you will see that there are in fact only three sprite definitions used for displaying the eight horses and by switching the block numbers each of the horses is animated and appears to be galloping.

Although we recommend three set areas for sprite definitions, you do not necessarily have to have the definitions running concurrently. You could if you wish have one definition in each area. It would be untidy programming but it would work.
Now you need to be able to move your spites around the screen and the next two SCOPE words enable you to do this.

3) **#DISPLAY;** *a number, a number or a Var, a number or a Var, a number or a Var:*

The first operand is the sprite number. (0-7)
The second operand is the colour number. (0-15)
The third operand is the Y or vertical co-ordinate. (0-200)
The fourth operand is the X or horizontal co-ordinate. (0-255)

Example:

10 REM #DISPLAY;0,5,100,100:

This display sprite 0 in green at co-ordinates 100,100.

10 REM VAR;A,100:
20 REM #DISPLAY;3,8,A,A:

This displays sprite 3 in orange at co-ordinates 100,100.

Note

If you have used Commodore sprites before you will remember that there are 200 vertical co-ordinates (Y) and 320 possible horizontal co-ordinates (X).

You can see that the word #DISPLAY only allows 255 of these possible X co-ordinates. To display at the other positions you must use a new SCOPE word:-

4) **#XMSB;** *a number:*

The operand is once again the sprite number combination taken from Table A. It tells the processor to display the requisite sprite or sprites at X co-ordinate 256 plus the X co-ordinate specified in the word #DISPLAY.

Each time this word is encountered it either sets or unsets the conditions explained above.

The following facts should be noted:-

A) The position of a sprite is calculated from the top left hand corner of the 24 buy 21 dot area that defines its shape, whether or not that part of the sprite is blank.
B) The first and last on screen values from the top to the bottom of the screen in the Y direction are
Fully on – 50
Fully off – 250
C) The first and last on screen values from left to right of the screen in the X direction are
Fully on – 24
Fully off – 343

Therefore to move a sprite fully across the middle of the screen in the X direction the following SCOPE program would be used.

To make things clear we give also a line by line explanation of the program.
First type and then run the following simple basic program to define a sprite at address 832.

```
10 FORA=832TO832+63
20 POKEA,255:NEXT
```
Then:-
```
1 REM NOTE; Routine to move a Sprite across the full width of the
screen:
10 REM ORG;15500,20000:
20 REM VAR;X,0:
30 REM VAR;Y,100:
35 REM SPRTS;1,0,0,0,0:
40 REM #BLK;0,13:
45 REM LABEL;A:
50 REM #DISPLAY;0,5,Y,X:
55 REM PAUSE;2:
60 REM INC;X,1:
70 REM TEST;X,255,240,A:
80 REM #XMSB;1:
90 REM VAR;X,0:
100 REM LABEL;B:
110 REM #DISPLAY;0,5,Y,X:
120 REM PAUSE;2:
130 REM INC;X,1:
140 REM TEST;X,88,240,B:
150 REM EXIT;
```
LINE 10 *Sets up the memory organisation.*
LINES 20-30 *Initialises variables X & Y for use as co-ordinates.*
LINE 35 *Enable Sprite 0.*
LINE 40 *Tells the processor that data for sprite 0 starts at memory location 832 (13*64).*
LINE 45 *Sets up a label called A.*
LINE 50 *Displays sprite 0 in colour green at co-ordinates X,Y.*
LINE 55 *Sets a pause.*
LINE 60 *Increases the value of X by 1.*
LINE 70 *Tests whether X equals 255 and if not jumps to label A. (Line 45)*
LINES 89-90 *When X does equal 255 this sets the screen position of X to 256 and the value of Var X to 0.*
LINE 100 *Sets up a label called B.*
LINE 110 *Displays sprite 0 in colour green at co-ordinates X,Y. Because #XMSB has been used this effectively sets the position of the sprite to 256. (256+0).*
LINE 120 *Sets a pause.*
LINE 130 *Increases the value of X by 1.*
LINE 140 *Test whether X=88 and if not jump to label B (Line 65). (Note that 256+88=343 to take the sprite fully off screen.)*
LINE 150 *Return to BASIC.*

Special Notes
You will usually want to position your sprites initially off screen and then smoothly move them into view. Remember therefore that you will have to use different starting positions for expanded sprites on either axis.

When you want to move an expanded sprite onto the left hand side of the screen in the X direction you must start your sprite off screen on the right hand side. This is because an expanded sprite is bigger than the amount of space available off screen on the left hand side.

Colours for sprites in Multi colour Mode
In this mode as in Graphics Multi colour mode data is interpreted in the following way.

Each dot in the sprite definition is twice as wide as normal. It is actually two pixels wide and the colour of each dot is determined by the pixel formation of these pairs as shown below.

Pixel Pair setting.

00	If both are blank then the screen background is used.
01	If the right hand pixel is set then the colour of Sprite multi colour register 1 is used.
10	If the left hand pixel is set then the sprite colour as specified in the word #DISPLAY is used.
11	If both pixels are set then the colour of sprite Multi colour register 2 is used.

Sprite multi colour register 1 is memory location 53285.
Sprite multi colour register 2 is memory location 53286.

To set these registers use the MEM word and the colour table as follows.

```
10 REM DVAR;A,53285:
20 REM DVAR;B,53286:
30 REM MEM;A,8:
40 REM MEM;B,14:
```

This sets Multi colour register 1 to orange and multi colour register 2 to Lt. blue.

Collision detection

There are two memory locations that contain data in respect of collisions. A collision occurs when a non zero part of a sprite overlaps a non zero portion of either another sprite or characters on the screen.

1) Memory location 53278 contains the data in respect of sprite to sprite collisions.
2) Memory location 53279 contains the data in respect of sprite to background collisions.

Each sprite has a bit in each register, so once again Table A is used, this time to interpret the data. The data in these registers, remain set until in SCOPE you use the word VIEW. Once 'viewed' the register is automatically cleared. VIEW of course saves the date read into a Var.

Example

```
10 REM DVAR;A,53278:
20 REM VIEW;A,B:
```

If Var;B contains 3 then sprite 0 has collided with sprite 1. Remember it takes two to make a collision!!!

```
10 REM DVAR;A,53279:
20 REM VIEW;A,B:
```

If Var;B now contained the value 3 then both sprites 0 & 1 have collided with background characters. If Var;b contained 1 then sprite 0 would have collided with background characters.

A last point to remember; collisions can take place off screen not only in the visible area, so you must be specific with your tests.

We now move on to the last section of SCOPE command words dealing with SOUND & MUSIC

Sound and Music

The COMMODORE 64 has extremely powerful sound capabilities which are fully explained in Chapter 4 of the Commodore Programmers Reference Guide.
SCOPE contains ten words dedicated to the handling of these facilities. We will go through each word first in detail with examples to make their use as clear as possible. The will follow some programming examples to give first hand experience of the types of sound that can be produced.

This manual is not the place for a detailed explanation of the way sound is received by the human ear, and how it can be simulated electronically. However, wherever necessary explanations have been given to try to make the various parameters understandable.

1) **#CLR;**

Whenever using the sound facilities one should always clear all the sound memory locations of any spurious data. The word #CLR does this for you.
You are going to exercise control over all the following parameters:-
A) The voice or voices selected (there are 3 possible.)
B) The waveform for each voice.
C) The envelope shape for each voice.
D) The volume level for all voices.
E) Any special effects for each voice.
F) The frequency or pitch of a note to be sounded.
G) The time that a note will sound.
H) The settings of the sound filter and which voice or voices will be routed through it.
I) The commencement of the sound envelope.
J) The completion of the sound envelope.
K) Advanced modulation techniques.

All this may look rather daunting but it is really fairly easy to make simple sounds and careful study will soon enable you to progress to more complex sound and three part music programming.

2) **#WAVE;** *a number or a Var, a number:*

The first operand is the waveform number.
The second operand is the voice number.

The voice numbers are 0,1, and 2.
The waveform numbers are as follows;

Triangle wave – 16.
Sawtooth wave – 32
Pulse wave – 64
White noise – 128

SPECIAL EFFECTS.
Synchronisation. – 2
Ring Modulation – 4

A triangle wave produces a flute like, hollow sound.
A sawtooth wave produces a more spacey, hollow sound.
A Pulse wave can produce a large range of sound tones according to the width of the pulse. See the word #PULSE for more details.
White noise produces rushing sounds which are more suitable for sound effects etc.

For further detailed explanation of waveforms refer to chapter 4 of the Commodore Programmers Reference Guide, or any good book on the elements of Sound.

Look at the following examples:-

10 REM #WAVE;10.0:

This sets the waveform for voice 0 to a triangle wave.

10 REM #WAVE;2,128:

This sets the waveform for voice 2 to white noise.

The word #WAVE however is also used to set up special effects. In addition to the waveform numbers you can add either one of the two effects numbers.

These effects are:

A) Synchronisation. This synchronises the base or fundamental frequency of one oscillator with the fundamental frequency of another, producing a wide range of harmonic structures.

B) Ring Modulation. This will replace the triangle waveform output of one oscilator with what is known as a ring modulated combination of that oscillator and one other. This is especially useful for gong sounds and other non-harmonic structures.

In both cases, which oscillator or voice is combined with which is controlled by the initial voice chosen in the #WAVE statement.

The rules for which voice combines with which are as follows;

1) If #WAVE statement specifies voice 0, then either of the effects will be with voice 3.
2) If #WAVE statement specifies voice 1 then either of the effects will be with voice 0.
3) If #WAVE statement specifies voice 2 then either of the effects will be with voice 1.

Examples:

10 REM #WAVE;34,0:

This will set a sawtooth waveform for voice 0 and synchronise it with the frequency chosen for voice 3.

10 REM #WAVE;20,1:

This will set a triangle waveform for voice 1 to be ring modulated with the output of voice 0.

The SCOPE word #WAVE used below specifies that voice 0 is set to a pulse wave.

10 REM #WAVE;64,0:

When using the pulse wave however you must also tell the processor what width of pulse you require, or you will get no sound. To do this you use the following word.

3) **#PULSE;** *a Dvar, a voice number:*

This word sets the pulse width for any chosen voice when the pulse waveform has been selected.

The pulse width number can vary between 1 and 4094. The number 2048 will produce a pulse width of equal proportions known as a square wave. Values of 0 or 4095 should not be used as they produce a constant D.C. output.

Study the following example;

```
10 REM DVAR;A,2048:
20 REM #WAVE;64,1:
30 REM #PULSE;A,1:
```

This sets voice 1 to a pulse wave and sets the pulse width to 2048, which is a square wave.

We now move on to the Envelope generator.

The volume level of a musical note or tone usually changes from the moment you first hear it to the moment that it becomes inaudible.

What happens is this.

When the note is first sounded it rises from zero volume until it reaches peak volume.
THE SPEED AT WHICH IT DOES THIS IS CALLED THE ATTACK

Next it falls from its peak volume to some middle range volume.

THE SPEED AT WHICH IT DOES THIS IS CALLED THE DECAY

The volume level it assumes after the DECAY is called the SUSTAIN level.

When the note stops playing it falls from its midrange level down to zero.
THE SPEED AT WHICH IT DOES THIS IS CALLED THE RELEASE.

These parameters are called for short A.D.S.R. and are all totally under the control of the programmer. It is these parameters that give a sound its own particular character.

The SCOPE word for handling these is:-

4) **#ADSR;***a voice number, a number, a number:*

The first operand is the voice number (0-2)
The second operand is the ATTACK/DECAY number.
The third operand is the SUSTAIN/RELEASE number.

The values are taken from Table B.

Table B

Value 1	*Attack Rate*	*Sustain Level*	*Value 2*	*Decay rate or Release rate*
0	2 ms	0	0	6 ms
16	8 ms	1	1	24 ms
32	16 ms	2	2	48 ms
48	24 ms	3	3	72 ms
64	38 ms	4	4	114 ms
80	56 ms	5	5	168 ms
96	68 ms	6	6	204 ms
112	80 ms	7	7	240 ms
128	100 ms	8	8	300 ms
144	250 ms	9	9	750 ms
160	500 ms	10	10	1.5 s
176	800 ms	11	11	2.4 s
192	1.0 s	12	12	3.0 s
208	3.0 s	13	13	9.0 s
224	5.0 s	14	14	15 s
240	8.0 s	15	15	24 s

Any combination of Value 1 + Value 2 make the ATTACK/DECAY number.

Any combination of Value 1 + Value 2 make the SUSTAIN/RELEASE number.

Example:

10 REM #ADSR;0,9,0:
This sets the ADSR for voice 0 to

An ATTACK rate of 2 ms.
A DECAY rate of 750 ms.
A SUSTAIN level of zero.
A RELEASE rate of 6 ms.

or
10 REM #ADSR;1,21,165:
This sets the ADSR for voice 1 to

An ATTACK rate of 8 ms.
A DECAY rate of 168 ms.
A SUSTAIN level of 10.
A RELEASE rate of 168 ms.

The next parameter to be set is the overall volume control for all three voices.

5)#**VOL**;a number or a Var:

For volume purposes only the number is a value from 0 to 15 where 0 is no volume and 15 is the maximum volume.

10 REM #VOL;15: Sets all voices to maximum volume.

or
10 REM VAR;A,15:
20 REM #VOL;A
This achieves the same.

The word #VOL however also handles some parameters of the programmable filter and one important parameter to do with special effects.

If you add 16 to your volume number you set the filter to LOW PASS.
If you add 32 to your volume number you set the filter to BANDPASS.
If you add 64 to your volume number you set the filter to HIGH PASS.
If you add 128 to your volume number you prevent voice 2 from having any audio output. This is useful for effects such as sync and ring modulation already mentioned and for special modulation effects. (See end of section).

Examples:

10 REM #VOL;31: Sets volume to maximum and filter to LOWPASS.

or
10 REM #VOL;143: Sets volume to maximum and disconnects voice 2 from audio output.

Before we go any further we had better briefly examine what sound filtering does and how it works.

Any note sounding at a given frequency does not only consist of that frequency but of weaker sounds above and below that frequency.

These are known as HARMONICS. Each harmonic bears a relation to the original or fundamental frequency. The first harmonic is twice the fundamental frequency, the second harmonic is three time the fundamental frequency and so on. The numbers of harmonics present in a note sounded are related to its waveform, and gives the note its own distinctive timbre.

FILTERING as the name suggests is a way of screening or eliminating some of these harmonics to change the overall harmonic content of a note and hence its timbre or sound.

A LOWPASS filter cuts down the level of harmonics *above* a specified frequency.
A HIGHPASS filter cuts down the level of harmonics *below* a specified frequency.
A BANDPASS filter passes a narrow band of harmonics each side of a specified frequency and cuts down the level of all others.

There is a further factor called RESONANCE.

It is a peaking effect which emphasises harmonics at the cut off frequency causing a sharper sound. There are 16 levels of resonance which can be chosen by the programmer.

As the programmer can also specify the cut off frequency, this gives enormous control over the final sound produced.

By the way you can combine LOWPASS and HIGHPASS filters to produce what is known as a NOTCH REJECT filter. This passes all harmonics except those at and around the cut off frequency.

The last thing to do then having chosen which filter is set, is to choose the cut off frequency and which voice or voices will be routed through it.

6) **#FILT;** *a number or a Var,a number or a Var:*

The first operand between 0-255 sets the cut off frequency of the filter. The second operand sets which voice or voices will be routed through and the level of resonance according to Table C.

The cut off frequency is roughly 47 times the number chosen. For instance a number of 100 will set the cut off frequency to 4700 hz and the maximum value of 255 will set the cut off frequency to 11985 hz. A value of 0 will give approx 30 hz giving an overall range of 30 – 12000 hz.

Table C

Value 1	*Effect*	*Value 2*	*Effect*
0	Low resonance	1	connect voice 0
16			to filter
32			
48			
64			
80			
96	Medium resonance	2	connects voice 1
112			to filter
128			
144			
160			
176		4	connects voice 2
192			to filter
208			
224			
240	High resonance		

Add the chosen Value 1 to the chosen Value 2 to arrive at the value for the 2nd operand of #FILT:

Example:

10 REM #FILT;10,2:
This sets a low cut off frequency, low resonance, and connects voice 1 to the filter

or

10 REM #FILT;240,244:
This sets a high cut off frequency, highest resonance and connects voice 2 to the filter.

So far we have gone into a lot of theory but haven't even found out how to sound a note yet.

There are two things left to deal with. How to choose a frequency or pitch for a note and how to sound a note, and tell it when to stop.

7) **#SOUND;** *a Dvar, a number:*
The first operand is the pitch number. See table D.
The second operand is the voice number.

Example:

10 REM DVAR A,4291:
20 REM #SOUND;A,0:
This sets voice 0 to the pitch of the note C-4 or middle C.

There are in fact two SCOPE words to select the pitch of a note. This one is called #SOUND because it gives very close resolution with no discernable stepping between frequences, enabling any type of sounds to be created. Gliding between notes, known as glissando or portamento, is easy to achieve. It can create excellent sound effects such as lasers, space guns, phasers etc., as well as music.

Writing music with this word only however, would be rather laborious and there is a special SCOPE word #MUSIC which combined with #SOUND makes the job much easier. This word will be explained at the end of this section.

To start a note sounding you should use the following SCOPE word.
8) **#GATE;** *a voice number:*

to finish a note sounding the word is
9) **#RLSE;** *a voice number.*

Examples:
10 REM #GATE;0:

or
10 REM #GATE;1:
20 REM #RLSE;1:

You will obviously have to use pauses in between these two words or you will never have time to hear the sound made. This will become apparent when we try some examples.

10) **#MUSIC;** *A label, a note number, a note number . . .*

This is a special kind of data statement. The note numbers are shown in table D. Each note number is read in turn and then the routine identified by the specified label is called to sound the note before returning for the next note. The colon (:) tells the processor that there are no further notes. For example:-

10 REM #MUSIC;A, 64,71,64,71,68,64:

I hope that this is self explanatory. Remember that the label used must refer to a routine or the system might crash!!!

Note
The word #MUSIC always applies to voice 0 only to enable a melody line to be written easily.

Hearing is believing, and we will shortly build up some sounds for you to hear in some demonstration programs.

Before we do however, there are two special registers which allow you to achieve special modulation techniques in real time.

MEMORY LOCATION 54299 – OSCILLATOR OUTPUT OF VOICE 2.
MEMORY LOCATION 54230 – ENVELOPE OUTPUT OF VOICE 2.

The first gives you a digital output of the voice 2 oscillator which is directly related to the waveform selected.
If you choose the triangle waveform the output will single step from 0 to 255 and then single step back down again to 0 and so on.
If you choose the sawtooth waveform the output will single step from 0 to 255 and then start again from 0 and so on.
If you choose the pulse waveform the output will jump back and forth giving 0 and then 255.
Finally if you choose the white noise waveform you will receive a series of random numbers between 0 – 255.

The second register gives you a similar digital output of the envelope shape chosen in numbers between 0–255.
When using these registers you do not want to hear any output from voice 2, so remember to disconnect it (See word #VOL:)

These registers can be 'VIEW'ed' and used to constantly alter;
a) the frequency of a note.
b) the volume of a note.
c) the envelope of a voice.
d) the cut off frequency of the filter.
etc., all in real time to achieve almost unlimited effects.
Let us now try some practical applications of all you have learnt so far.

Musical Note Values
The musical note values are shown in table D. The values under the heading NOTE are the values for the word #MUSIC and the values under heading DECIMAL are the values for the word #SOUND.

Table D

MUSICAL NOTE		DECIMAL	MUSICAL NOTE		DECIMAL
NOTE	OCTAVE		NOTE	OCTAVE	
0	C–0	268	64	C–4	4291
1	C#–0	284	65	C#–4	4547
2	D–0	301	66	D–4	4817
3	D#–0	318	67	D#–4	5103
4	E–0	337	68	E–4	5407
5	F–0	358	69	F–4	5728
6	F#–0	379	70	F#–4	6069
7	G–0	401	71	G–4	6430
8	G#–0	425	72	G#–4	6812
9	A–0	451	73	A–4	7217
10	A#–0	477	74	A#–4	7647
11	B–0	506	75	B–4	8101
16	C–1	536	80	C–5	8583
17	C#–1	568	81	C#–5	9094
18	D–1	602	82	D–5	9634
19	D#–1	637	83	D#–5	10207
20	E–1	675	84	E–5	10814
21	F–1	716	85	F–5	11457
22	F#–1	758	86	F#–5	12139
23	G–1	803	87	G–5	12860
24	G#–1	851	88	G#–5	13625
25	A–1	902	89	A–5	14435
26	A#–1	955	90	A#–5	15294
27	B–1	1012	91	B–5	16203
32	C–2	1072	96	C–6	17167
33	C#–2	1136	97	C#–6	18188
34	D–2	1204	98	D–6	19269
35	D#–2	1275	99	D#–6	20415
36	E–2	1351	100	E–6	21629
37	F–2	1432	101	F–6	22915
38	F#–2	1517	102	F#–6	24278
39	G–2	1607	103	G–6	25721
40	G#–2	1703	104	G#–6	27251
41	A–2	1804	105	A–6	28871
42	A#–2	1911	106	A#–6	30588
43	B–2	2025	107	B–6	32407
48	C–3	2145	112	C–7	34334
49	C#–3	2273	113	C#–7	36376
50	D–3	2408	114	D–7	38539
51	D#–3	2551	115	D#–7	40830
52	E–3	2703	116	E–7	43258
53	F–3	2864	117	F–7	45830
54	F#–3	3034	118	F#–7	48556
55	G–3	3215	119	G–7	51443
56	G#–3	3406	120	G#–7	54502
57	A–3	3608	121	A–7	57743
58	A#–3	3823	122	A#–7	61176
59	B–3	4050	123	B–7	64814

Examples of the Use of Sound

Type in the following:-

```
10 REM ORG;15500,20000:
20 REM DVAR;A,4291:
30 REM #WAVE;16,0:
40 REM #ADSR;0,9,0:
50 REM #VOL;15:
60 REM #SOUND A,0:
70 REM #GATE;0:
80 REM PAUSE;250:
90 REM #RLSE;0:
95 REM EXIT;
```

Now compile it (SYS 49583)
and run it (SYS 15500).

At last you have heard a sound!

Now alter line 30 to

```
30 REM #WAVE;32,0:
```

Compile it and run it again and you will notice the sound is much sharper and rougher.

Change line 30 again.

```
30 REM #WAVE;128,0:
```

Compile and run. Now you have the distinctive sound of white noise.

Finally change line 30 and add two extra lines as below.

```
15 REM DVAR;B,2148:
30 REM #WAVE;64,0:
35 REM #PULSE;B,0:
```

The sound has changed again and you have now produced a square wave.

So much for the wave form; delete lines 15 and 35 and change lines 30 and 40 as follows.

```
30 REM #WAVE;16,0:
40 REM #ADSR;0,0,246:
```

Try altering the #ADSR values until you understand what is happening.

Your BASIC manual gives you some suggested settings to simulate various instruments.

After you have done that try the following;

```
10 REM ORG; 15500,20000;
20 REM DVAR;A,100;
30 REM #WAVE;32,0;
40 REM #ADSR;0,0,240;
50 REM #VOL;15;
60 REM LABEL;B;
65 REM #SOUND;A,0;
70 REM #GATE;0;
80 REM ADD;A,20;
85 REM GET;K;
90 REM TEST;K,65,240,B;
95 REM #CLR;
100 EXIT;
```

The sound will continue to cycle until you press key 'A'.

There follows some other listings for you to try.

It is suggested that you work your way through the various SCOPE words for Sound and Music until you are familiar with how they work and can produce the level of sound that you wish to achieve. It is hoped that the demonstration programs given on the reverse side of the cassette tape, which are also listed in the special section in this manual will act as guide lines, together with the other listings given.

```
1 REM NOTE;EXAMPLE OF RING MODULATION:
5 REM NOTE;PRESS KEY 'A' TO END:
10 REM ORG;15500,20000:
15 REM #CLR:
18 REM 1 LABEL;Z:
20 REM DVAR;A,4800:
25 REM DVAR;B,3000:
30 REM #WAVE;20,0:
35 REM #WAVE;16,2:
38 REM #SOUND;B,2:
40 REM #GATE;2:
50 REM #ADSR;0,0,249:
55 REM #VOL;143:
58 REM #SOUND;A,0:
60 REM #GATE;0:
65 REM PAUSE;150:
70 REM #RLSE;0:
75 REM PAUSE;250:
76 REM PAUSE;250:
77 REM PAUSE;250:
78 REM PAUSE;250:
80 REM #CLR:
85 REM GET;K:
90 REM TEST;K,65,240,Z:
200 REM EXIT;
```

```
1 REM NOTE;MODULATION EFFECT – SPACE SHIP SOUND?:
5 REM NOTE;PRESS KEY 'A' TO END:
10 REM ORG;15500,20000:
15 REM DVAR;A,3500:
18 REM DVAR;D,54273:
20 REM #WAVE;16,0:
22 REM #WAVE;16,2:
25 REM #ADSR;0,0,240:
30 REM DVAR;C,10:
32 REM #SOUND;C,2:
35 REM #GATE;2:
40 REM DVAR;B,54229:
42 REM #VOL;143:
45 REM LABEL;A:
50 REM VIEW;B,A:
55 REM MEM;D,A:
57 REM PAUSE;2:
60 REM #SOUND;A,0:
65 REM ADD;A,10:
68 REM #GATE;0:
70 REM GET;K:
75 REM TEST;K,65,240,A:
80 REM #CLR;
200 REM EXIT;
```

```
5 REM NOTE;CAR ENGINE?:
10 REM ORG;15500,20000:
15 REM #CLR;
20 REM DVAR;A,1000:
25 REM DVAR;B,2048:
27 REM #WAVE;16,1:
30 REM #WAVE;32,0:
32 REM #PULSE;B,0:
35 REM #ADSR;0,0,240:
36 REM #ADSR;0,0,240:
38 REM #VOL;15:
40 REM RTN;
45 REM LABEL;A:
48 REM #GATE;0:
50 REM #GATE;1:
55 REM #SOUND;A,0:
56 REM #SOUND;A,1:
60 REM #PAUSE;10:
62 REM ADD;A,255:
65 REM #SOUND;A,0:
66 REM #SOUND;A,1:
67 REM MINUS;A,255:
68 REM PAUSE;10:
80 REM END;
81 REM VAR;A,0:
82 REM LABEL;B:
85 REM CALL;A:
87 REM ADD;A,10:
90 REM INC;A,1:
95 REM TEST;A,255,240,R:
98 REM VAR;A,0:
100 REM LABEL;C:
105 REM CALL;A:
110 REM INC;A,1:
115 REM TEST;A,255,240,C:
120 REM VAR;A,0:
125 REM LABEL;D:
130 REM CALL;A:
135 REM MINUS;A,15:
140 REM INC;A,1:
145 REM TEST;A,255,240,D:
1999 REM #CLR;
2000 REM EXIT;
```

Using Scope from BASIC and with other Machine Code Routines

SCOPE can be used as a complete language for writing finished games and other programs. There may be occasions however, when it is preferred to call SCOPE routines from within a BASIC program. Also you may wish during the course of a SCOPE program to utilise machine code routines you have written yourself.

TO CALL A SCOPE ROUTINE FROM BASIC

First write and complete your SCOPE routine making a note of where it has been compiled. You can then call the machine code routine that SCOPE has written by means of a SYS call.

If you have more than one routine, and you have used any of the three special words then you need only save from 15500 the first time, so that the others can be in higher memory if you wish.

You may wish to transfer BASIC variables into SCOPE variables. In the appendices you will find the memory addresses of all SCOPE variables and double variables. From BASIC you simply poke your BASIC variable values into the appropriate SCOPE variable addresses, and then call your SCOPE routine.

TO CALL A MACHINE CODE ROUTINE FROM SCOPE

In the appendices there is a list of the memory addresses used for labels. Choose a label that you wish to designate for your machine code routine and poke the address of your machine code routine into the table for that label. The values should be poked low byte/high byte.

Then whenever you CALL that label (NOT JUMP) you will be calling your machine code routine.

Some Useful Addresses

SCOPE SYSTEM VARIABLES

LABEL TABLE	40500 – 40707	A – Z	40500 – 40551
	In the order:-	#A – #Z	40552 – 40603
	Normal, #, $, %.	$A – $Z	40604 – 40655
		%A – %Z	40656 – 40707
VARIABLE TABLE	49308 – 49359	A – Z	49308 – 49333
	In the order:-	#A – #Z	49334 – 49359
	Normal, #.		
DVAR TABLE	49366 – 49417		
N.A.P. *(Next Address Pointer)*	251 – 252		
R.A.P. *(Routine Address Pointer)*	49522 – 3		

Some Useful Routines

1) **A Division Routine**
2) **A Multiplication Routine**
3) **A Routine to copy the character set from ROM into RAM at 12288.**

```
5 REM NOTE;ROUTINE TO DIVIDE (X/Y):
10 REM ORG;15500,20000:
20 REM LABEL;A:
30 REM DVAR;A,0:
40 REM VAR;X,7:
50 REM LABEL;A:
60 REM DVAR;A,0:
70 REM VAR;X,7:
75 REM NOTE;X/Y:
80 REM VAR;Y,5:
90 REM LABEL;C:
100 REM DEC;X,Y:
110 REM TEST;X,200,144,B:
120 REM ADD;A,1:
130 REM JUMP;C:
135 REM NOTE;X/Y:
140 REM LABEL;B:
150 REM NUM;10,10,1,A:
160 REM EXIT;
```

```
5 REM NOTE;ROUTINE TO MULTIPLY (X*Y)
10 REM ORG;15500,20000:
20 REM CALL;A:
70 REM RTN;
80 REM LABEL;A:
90 REM DVAR;A,0:
100 REM VAR;Y,255:
110 REM ADD;A,Y:
120 REM VAR;X,255:
130 REM LABEL;R:
140 REM ADD;A,Y:
150 REM DEC;X,1:
160 REM TEST;X,1,240,B:
170 REM PUT;10,0,1,"RESULT=":
180 REM NUM;10,8,1,A:
190 REM END;
200 REM EXIT;
```

```
1 REM NOTE;ROUTINE TO COPY CHARACTER SET FROM ROM TO
MEMORY LOCATION 12288:
5 REM NOTE;IT ALSO SETS THE POINTERS TO THE NEW LOCATION
OF THE CHARACTER SET:

10 REM ORG;15500,20000:
20 REM ALTER;12000:
30 REM DVAR;B,56334:
40 REM MEM;B,0:
50 REM DVAR;C,1:
60 REM MEM;C,51:
70 REM DVAR;E,53248:
80 REM DVAR;D,12288:
90 REM LABEL;A:
100 REM VIEW;F,A:
110 REM MEM;D,A:
120 REM ADD;E,1:
130 REM ADD;D,1:
140 REM LIM;E,55288,B:
150 REM JUMP;A:
160 REM LABEL;B:
170 REM MEM;C,55:
180 REM MEM;B,1:
190 REM DVAR;F,53272:
195 REM MEM;F,29:
200 REM EXIT;
```

Dictionary

Word	*Format*	*Usage*
ADD	*A Dvar, a number/Var:*	Increases a Dvar.
ALTER	*a number:*	Sets the top of memory.
BDR	*a number/Var:*	Sets the border colour.
BACK	*a number/Var:*	Sets the background colour.
COL	*a number, a number, a number:*	Sets the colours for Mode 1
CALL	*a label:*	Calls a routine.
DEC	*a Var, a number/Var:*	Decrements a variable.
DVAR	*a letter, a number:*	Sets the value of a Dvar.
END	;	Ends a routine.
EXIT	;	Returns to BASIC.
GET	*a variable:*	Reads the keyboard.
GMODE	*a number:*	Sets the Graphics Mode.
INC	*a variable, a number/Var:*	Increases a variable.
JUMP	*a label:*	Jumps to a label.
LIM	*a Dvar, a number, a Label:*	Tests a Dvar and executes a jump if condition satisfied.
LABEL	*a letter:*	Sets a Label.
MEM	*a Dvar, a number/Var:*	Pokes a value into a memory address.
MINUS	*a Dvar, a number/Var:*	Decreases a Dvar.
NUM	*a number, a number, a number/Var, a Dvar:*	Prints a number at specified coordinates. in specified colour.
NOTE	*Text:*	Text is ignored by processor.
ORG	*a number, a number:*	Sets the memory organisation.
PUT	*a number/Var, a number/Var, a number/Var, "Text":*	Prints characters at coordinates specified. in the colour specified.
PAUSE	*a number/Var:*	Sets a pause.
RTN	*a letter:*	Initiates a routine.
RAND	*a Var, a number, a number:*	Sets a random number into a Var.
SCR	*a number:*	Scrolls the screen in any 4 directions.
SPRTS	*a number, a number, a number, a number, a number:*	Initialises the sprite parameters.
TCO	*a number,a number, a number, a number:*	Sets up a split colour screen.

TEST	*a variable, a number/Var , a test number, a label:*	Test the value of a Var for a specified condition and jumps if satisfied.
VAR	*a letter, a number/Var:*	Sets up a Variable.
VIEW	*a Dvar, a number/Var:*	Puts the contents of a memory location into a variable.
WIPE	*;*	Clears the screen.
XBDR	*a number:*	Shrinks the screen as chosen.
#ADSR	*a number, a number, a number:*	Sets the A.D.S.R. parameters of a sound.
#BLK	*a number, a number:*	Sets the address of sprite data.
#CLR	*;*	Clears all sound registers.
#DISPLAY	*a number, a number/Var, a number/Var, a number/Var:*	Puts a sprite on the screen at the coordinates and colour specified.
#FILT	*a number/Var, a number/Var:*	Sets the parameters of the sound filter.
#GATE	*a number:*	Starts the sound envelope.
#MUSIC	*a label, a note number, . . .:*	Sets the pitch of a note and calls routine to sound it and returns for the next number in the statement.
#PULSE	*a Dvar, a voice number:*	Sets the pulse width of a note.
#RLSE	*a number:*	Starts the end of the sound envelope.
#SOUND	*a Dvar, a number:*	Sets the pitch of a note.
#VOL	*a number/Var:*	Sets the volume and effects of a note.
#WAVE	*a number, a number:*	Sets the waveform of a note.
#XMSB	*a number:*	Sets the R/H screen position of a sprite.

Listings of the Demonstration Routines on the other side of your Scope cassette.

```
5 REM NOTE;DEMONSTRATION ROUTINE 'SHOOT IN':
10 REM ORG;15500,16500:
20 REM #CLR;
25 REM LABEL;Z:
30 REM VAR;X,25:
40 REM VAR;Y,15:
50 REM BDR;4:
60 REM BACK;5:
70 REM WIPE;
80 REM #VOL;15:
81 REM #WAVE;128,0:
82 REM #ADSR;0,0,160:
83 REM DVAR;F,500:
90 REM TCO;80,182,2,5:
100 REM CALL;A:
110 REM PUT;X,Y,6,"S":
120 REM INC;Y,1:
130 REM VAR;X,25:
140 REM CALL;A:
150 REM PUT;X,Y,6,"C":
160 REM INC;Y,1:
170 REM VAR;X,25:
180 REM CALL;A:
185 REM PUT;X,Y,6,"O":
190 REM INC;Y,1:
200 REM VAR;X,25:
205 REM CALL;A:
210 REM PUT;X,Y,6,"P":
220 REM INC;Y,1:
230 REM VAR;X,25:
240 REM CALL;A:
250 REM PUT;X,Y,6,"E":
300 REM PUT;9,5,0,"COMPUTER":
310 REM PUT;9,14,6,"GRAPHICS":
320 REM PUT;9,23,0,"LANGUAGE":
325 REM DVAR;F,12500:
327 REM #WAVE;32,0:
350 REM VAR;C,0:
360 REM LABEL;C:
365 REM #SOUND;F,0:
368 REM #GATE;0:
370 REM SCR;3:
375 REM MINUS;F,75:
380 REM INC;C,1:
385 REM PAUSE;5:
390 REM TEST;C,20,240,C:
400 REM #WAVE;128,0:
410 REM #ADSR;0,9,172:
420 REM DVAR;F,1500:
```

```
430 REM #SOUND;F,0:
440 REM #GATE;0:
450 REM PAUSE;250:
460 REM #RLSE;0:
610 REM PAUSE;255:
620 REM PAUSE;255:
1000 REM RTN;
1010 REM LABEL;A:
1015 REM #SOUND;F,0:
1017 REM #GATE;0:
1020 REM PUT;X,Y,0," ":
1030 REM DFC;X,1:
1040 REM PUT;X,Y,1," ↑ ":
1042 REM ADD;F,100:
1045 REM PAUSE;5:
1050 REM TEST;X,5,240,A:
1055 REM CALL;B:
1056 REM #RLSE;0:
1057 REM DVAR;F,500:
1060 REM END;
2000 REM RTN;
2010 REM LABEL;B:
2012 REM DVAR;C,45000:
2015 REM #SOUND;C,0:
2016 REM #GATE;0:
2020 REM PUT;X,Y,4,"X":
2030 REM PAUSE;100:
2040 REM PUT;X,Y,4,"+":
2050 REM PAUSE;100:
2055 REM #RLSE;0:
2060 REM END;
2065 REM UCO;
2070 REM JUMP;Z:
9999 REM #CLR;
10000 REM EXIT;
```

```
5 REM NOTE;DEMONSTRATION ROUTINE 'WMUSIC':
10 REM ORG;15500,20000:
11 REM #CLR;
12 REM #WAVE;16,2:
13 REM XBDR;1:
14 REM WIPE;
15 REM #WAVE;16,0:
16 REM VAR;Q,0:
17 REM #ADSR;2,0,240:
18 REM #WAVE;16,1:
19 REM #ADSR;1,12,172:
20 REM #ADSR;0,1:45:
21 REM DVAR;Y,1432:
22 REM #SOUND;Y,2:
23 REM #GATE;2:
25 REM #VOL;15:
27 REM RTN;
30 REM LABEL;A:
40 REM #GATE;0:
45 REM PAUSE;20:
50 REM #RLSE;0:
52 REM PAUSE;20:
55 REM END;
56 REM RTN;
58 REM LABEL;B:
60 REM #MUSIC;A,48,50,53,57:
65 REM #MUSIC;A,64,66,69,73,80,82,85,89:
70 REM #MUSIC;A,96,89,85,82,80,73,69,66:
75 REM #MUSIC;A,64,57,53,50,48,41:
80 REM END;
82 REM RTN;
85 REM LABEL;C:
90 REM DVAR;A,2145:
92 REM CALL;D:
93 REM CALL;B:
94 REM CALL;J:
95 REM DVAR;A,2408:
96 REM CALL;D:
98 REM CALL;B:
99 REM CALL;J:
100 REM DVAR;A,3608:
105 REM CALL;D:
110 REM CALL;B:
112 REM CALL;J:
115 REM DVAR;A,2864:
120 REM CALL;D:
125 REM CALL;B:
127 REM CALL;J:
130 REM INC;A,1:
135 REM TEST;A,2,240,C:
140 REM END;
160 REM RTN;
162 REM LABEL;D:
```

```
165 REM #SOUND;A,1:
168 REM #GATE;1:
170 REM PAUSE,25:
172 REM #RLSE;1:
175 REM END;
177 REM VAR;A,0:
180 REM CALL;C:
185 REM #WAVE;32,1:
187 REM VAR;A,0:
190 REM CALL;C:
195 REM #WAVE;64,1:
198 REM DVAR;7,2048:
200 REM #PULSE;Z,1:
201 REM #VOL;5:
202 REM VAR;A,0:
205 REM CALL;C:
207 REM #WAVE;32,1:
210 REM #VOL;2:
215 REM VAR;A,0:
220 REM CALL;C:
225 REM #CLR;
230 REM RTN;
231 REM LABEL;J:
234 REM BACK;Q:
235 REM BDR;Q:
238 REM PUT;24,11,4,"  S C O P E  ":
240 REM SCR;2:
241 REM PUT;24,11,5,"COMPUTER GRAPHICS":
242 REM SCR;2:
243 REM PUT;24,11,7,"  LANGUAGE  ":
244 REM SCR;2:
245 REM PUT;24,11,7,"            ":
246 REM SCR;2:
248 REM INC;Q,1:
260 REM END;
9999 REM EXIT;
```

```
1 REM NOTE;DEMONSTRATION ROUTINE 'ASCOT':
5 REM ORG;15500,20000:
6 REM #WAVE;128,0:
7 REM #ADSR;0,1,161:
8 REM #VOL;15:
9 REM DVAR;A,15:
10 REM #SOUND;A,0:
12 REM WIPE;
20 REM VAR;A,255:
25 REM VAR;B,A:
30 REM VAR;C,A:
32 REM VAR;D,A:
35 REM VAR;E,A:
38 REM VAR;F,A:
40 REM VAR;G,A:
42 REM VAR;H,A:
45 REM BDR;0:
48 REM BACK;5:
62 REM SPRTS;255,0,0,0,65:
65 REM VIEW A,J:
68 REM LABEL;A:
70 REM #DISPLAY;0,0,58,A:
72 REM #DISPLAY;1,9,80,B:
75 REM #DISPLAY;2,6,102,C:
78 REM #DISPLAY;3,3,124,D:
80 REM #DISPLAY;4,1,146,E:
82 REM #DISPLAY;5,11,168,F:
85 REM #DISPLAY;6,15,190,G:
87 REM #DISPLAY;7,2,212,H:
90 REM #BLK;0,13:
92 REM #BLK;1,14:
95 REM #BLK;2,15:
98 REM #BLK;3,13:
100 REM #BLK;4,14:
102 REM #BLK;5,15:
105 REM #BLK;6,13:
108 REM #BLK;7,14:
109 REM CALL;D:
112 REM #BLK;0,14:
115 REM #BLK;1,15:
118 REM #BLK;2,13:
119 REM #BLK;3,14:
120 REM #BLK;4,15:
122 REM #BLK;5,13:
125 REM #BLK;6,14:
128 REM #BLK;7,15:
129 REM CALL;D:
132 REM #BLK;0,15:
135 REM #BLK;1,13:
138 REM #BLK;2,14:
140 REM #BLK;3,15:
142 REM #BLK;4,13:
145 REM #BLK;5,14:
148 REM #BLK;6,15:
150 REM #BLK;7,13:
```

```
151 REM CALL;D:
155 REM RAND;L,6,0:
157 REM DEC;A,L:
158 REM TEST;A,24,176,C:
159 REM RAND;M,6,0:
160 REM DEC;B,M:
161 REM TEST;B,24,176,C:
162 REM RAND;N,6,0:
165 REM DEC;C,N:
166 REM TEST;C,24,176,C:
168 REM RAND;O,6,0:
170 REM DEC;D,O:
171 REM TEST;D,24,176,C:
172 REM RAND;P,6,0:
175 REM DEC;E,P:
176 REM TEST;E,24,176,C:
178 REM RAND;Q,6,0:
180 REM DEC;F,Q:
181 REM TEST;F,24,176,C:
182 REM RAND;R,6,0:
185 REM DEC;G,R:
187 REM TEST;G,24,176,C:
188 REM RAND;S,6,0:
190 REM DEC;H,S:
191 REM TEST;H,1,176,C:
195 REM JUMP;A:
196 REM LABEL;C:
197 REM #ADSR;0,2,69:
198 REM #WAVE;16,0:
199 REM DVAR;A,8500:
200 REM #SOUND;A,0:
201 REM #GATE;0:
202 REM PAUSE;150:
203 REM #RLSE;0:
204 REM PAUSE;255:
205 REM #CLR;
206 REM PUT;23,10,6,"RACE OVER":
209 REM RTN;
210 REM LABEL;D:
211 REM #GATE;0:
212 REM PUT;0,0,0,"O":
215 REM PUT;24,0,0,"O":
216 REM #RLSE;0:
220 REM SCR;0:
222 REM PUT;0,0,0,"-":
223 REM #GATE;0:
225 REM PUT;24,0,0,"-":
230 REM SCR;0:
232 REM PUT;0,0,0,"-":
234 REM PUT;24,0,0,"-":
235 REM SCR;0:
236 REM #RLSE;0:
240 REM END;
9999 REM EXIT;
```

```
5 REM NOTE;DEMONSTRATION ROUTINE 'MOZART':
10 REM ORG;15500,20000:
12 REM #CLR;
13 REM DVAR;C,6430:
14 REM #WAVE;16,2:
15 REM #WAVE;16,0:
16 REM #WAVE;16,1:
17 REM DVAR;B,2148:
18 REM DVAR;E,3000:
19 REM DVAR;D,4000:
20 REM #ADSR;0,0,164:
21 REM #ADSR;2,0,240:
22 REM #ADSR;1,0,240:
23 REM #GATE;1:
24 REM #GATE;2:
25 REM #VOL;15:
28 REM RTN;
30 REM LABEL;A:
32 REM #SOUND;A,1:
33 REM #SOUND;C,2:
35 REM #GATE;0:
40 REM PAUSE;40:
45 REM #RLSE;0:
50 REM PAUSE;100:
55 REM END;
56 REM LABEL;Z:
57 REM DVAR;C,6430:
58 REM DVAR;A,1072:
60 REM #MUSIC;A,64,68,71,64,68,71:
61 REM DVAR;C,5728:
62 REM DVAR;A,1607:
65 REM #MUSIC;A,59,66,71,59,66,71:
66 REM DVAR;C,4817:
67 REM DVAR;A,803:
70 REM #MUSIC;A,59,66,71,59,66,71:
71 REM DVAR;C.5407:
72 REM DVAR;A,1072:
75 REM #MUSIC;A,64,68,71,64,68,71:
76 REM DVAR;C,8583:
80 REM #MUSIC;A,64,68,71,64,68,71:
81 REM DVAR;C,5728:
82 REM DVAR;A,1432:
85 REM #MUSIC;A,64,69,73,64,69,73:
86 REM DVAR;C,6430:
87 REM DVAR;A,803:
90 REM #MUSIC;A,64,68,71,55,57,59:
91 REM DVAR;C,8583:
92 REM DVAR;A,536:
95 REM #MUSIC;A,64,68,71,80:
100 REM PAUSE;150:
105 REM #WAVE;64,0:
110 REM #PULSE;B,0:
115 REM #WAVE;64,1:
```

```
120 REM #PULSE;D,1:
125 REM #WAVE;64,2:
127 REM #GATE;1:
130 REM #PULSE;E,2:
132 REM #GATE,2:
135 REM GET;K:
140 REM TEST;K,65,240,Z:
199 REM #CLR;
200 REM EXIT;
```

```
5 REM NOTE;DEMONSTRATION ROUTINE 'BACH':
10 REM ORG;15500,20000:
12 REM #CLR;
15 REM #WAVE;16,0:
20 REM #WAVE;16,0:
25 REM #ADSR;0,128:
30 REM #ADSR;1,0,240:
35 REM #VOL;15:
40 REM RTN;
45 REM LABEL;A:
46 REM #SOUND;A,1:
47 REM #GATE;1:
50 REM #GATE;0:
55 REM PAUSE;60:
60 REM #RLSE;0:
65 REM PAUSE;50:
70 REM END;
90 REM RTN;
95 REM LABEL;B:
110 REM DVAR;A,1072:
120 REM #MUSIC;A,68,66,68,69:
125 REM MINUS;A,60:
135 REM #MUSIC;A,71,68,69,71:
140 REM MINUS;A,110:
150 REM #MUSIC;A,64,59,64,66:
155 REM MINUS;A,99:
165 REM #MUSIC;A,68,69,71,68:
170 REM DVAR;A,716:
180 REM #MUSIC;A,73,71,73,74:
185 REM ADD;A,42:
195 REM #MUSIC;A,73,71,69,68:
200 REM ADD;A,45:
210 REM #MUSIC;A,66,80,75,73,71,69,68,66:
215 REM #RLSE;1:
220 REM DVAR;A,1072:
225 REM #MUSIC;A,68,66,68,69:
230 REM MINUS;A,60:
235 REM #MUSIC;A,71,68,69,71:
240 REM MINUS;A,110:
245 REM #MUSIC;A,64,59,64,66:
250 REM MINUS;A,99:
255 REM #MUSIC;A,68,64,66,68:
260 REM DVAR;A,716:
265 REM #MUSIC;A,57,59,64,69:
270 REM ADD;A,87:
275 REM #MUSIC;A,68,68,66,66:
280 REM DVAR;A,536:
282 REM DVAR;B,4291:
283 REM #SOUND;B,0:
284 REM #SOUND;A,1:
285 REM #GATE;0:
286 REM #GATE;1:
290 REM PAUSE;200:
```

```
292 REM PAUSE;200:
293 REM PAUSE;200:
295 REM #RLSE;1:
296 REM #RLSE;0:
298 REM END;
300 REM CALL;B:
305 REM #WAVE;32,0:
306 REM #WAVE;32,1:
310 REM CALL;B:
315 REM DVAR;C,2048:
320 REM #PULSE;C,0:
326 REM #WAVE;64,0:
328 REM #VOL;6:
330 REM CALL;B:
340 REM #CLR;
999 REM EXIT;
```

```
1 REM NOTE;DEMONSTRATION ROUTINE 'MOSAIC':
5 REM ORG;15500,20000:
6 REM WIPE;
10 REM LABEL;$B:
12 REM VAR;A,0:
13 REM BDR;D:
14 REM BACK,0:
15 REM LABEL;%A:
20 REM RAND;B,40,0:
25 REM RAND;C,24,0:
27 REM RAND;D,15,1:
30 REM PUT;C,B,D,"♥":
32 REM INC;A,1:
35 REM TEST;A,255,240,%A:
40 REM GET;K:
45 REM TEST;K,65,240,$B:
200 REM EXIT;
```

```
5 REM NOTE;DEMONSTRATION ROUTINE 'SPACE SOUND':
10 REM ORG;15500,20000:
15 REM DVAR;B,35:
20 REM DVAR;A,500:
30 REM #WAVE;128,0:
35 REM #ADSR;0,9,169:
40 REM #VOL;15:
45 REM #SOUND;A,0:
48 REM #SOUND;B,2:
50 REM #GATE;0:
52 REM LABEL;B:
53 REM #SOUND;A,0:
55 REM ADD;A,20:
56 REM ADD;B,20:
57 REM HALT;10:
60 REM VAR;A,0:
65 REM GET;A:
70 REM TEST;A,65,240,B,0:
75 REM #RLSE;0:
200 REM EXIT;
```

SCOPE USER'S CLUB

Application for Membership

By simply completing this sheet and sending it to us you will be able to benefit from the unique **Scope User's Club.** The aim of the club is to provide a forum for users and also a definitive source of information and assistance for the serious young programmer.

On receipt of your application form we will send you a membership package which will include your personalised membership card upon which will be the programmer's telephone hot-line number.

Additionally, we hope to be producing a regular news-sheet to keep you informed of developments and details of special offers.

Complete this page, tear it out and send it to us to join a really unique club.

APPLICANT NAME _____

ADDRESS _____

_____

_____

PHONE _____ AGE _____

COMPUTERS OWNED _____

WHERE DID YOU FIRST LEARN OF SCOPE? *Please tick as applicable.*

☐ FRIEND ☐ RETAILER ☐ PUBLICATION

IF PUBLICATION, PLEASE STATE WHICH ONE _____

WHAT COMPUTER PUBLICATIONS DO YOU READ REGULARLY?

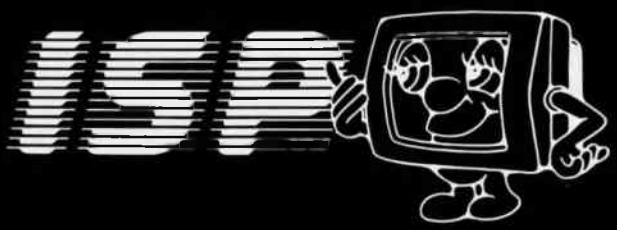
_____

_____

WHERE DID YOU BUY YOUR COPY OF SCOPE? *Please tick as applicable.*

☐ MAILORDER ☐ RETAILER

NAME OF RETAILER _____

_____

Please post your completed form to:
ISP Marketing Ltd., 15a Castons Yard, Basingstoke, Hants.